SQL-based Approach for Compiling Ordered Decision Diagrams from Device Models

Yousri El Fattah
Rockwell Science Center
1049 Camino Dos Rios
Thousand Oaks, CA 91360
yelfattah@rwsc.com

October 15, 2005

Abstract

To enable autonomy and fault-tolerance intelligent software is needed that takes a device model and compiles it into structure that facilitates computing in real-time a systems operating mode and how to reconfigure to a desired target mode. We describe an approach that compiles a relational database representation of a system description into a structured procedure (SP) for diagnosis. The approach uses variable elimination to compute the SP in a compilation phase, prior to diagnosis, in the form of a directed acyclic graph (DAG). The nodes of the dag are SQL queries and the edges specify query execution precedence. The SP can be computed from the system structure and the subset of observable variables and can exploit the non-structural properties to simplify the elimination steps. At diagnosis time the SP is used to perform the diagnosis task, taking as input the values of the observed variables and producing as output the set of all minimal diagnoses.

1 Introduction

Future Moon and Mars explorations will depend on cooperative space systems that operate largely autonomously without ground support for extended periods of time. In order to enable this capability, such a system of systems will require a new generation of robust, fault-tolerant software that

supports autonomous operation. The use of such intelligent software would lead to improved sustainability of exploration missions by enhanced effectiveness and efficiency through appropriate responses to anomalous events, increased safety and reliability through quick and autonomous responses to faults, increased affordability through reduced cost of ground operations, and increased flexibility through applicability to a wide variety of mission systems and platforms.

To enable autonomy and fault-tolerance intelligent software is needed that takes a device model and compiles it into structure that facilitates computing in real-time a systems operating mode and how to reconfigure to a desired target mode [Barrett2005]. Traditional approach relies on hand-crafting rule-based diagnosis and recovery systems that require intensive human effort and expertise at highly increasing cost. As complexity spirally increases there is the inevitability that humans will overlook subtle interactions of how components can interact and fail producing rule-based systems with limited coverage of the space of possible faults. Furthermore the hand-crafted rule-based systems are hard to revise for consistency with change in device models and the rule-based inference engine may be ad-hoc making it hard to provide guarantees on soundness and on computational complexity.

To address the need for automated approaches to compiling diagnostic rules from device models few approaches have been proposed that rely on formulating device models in propositional logic and the computing of a compiled representation in the form of prime implicates [J.de Kleer1990] or decomposable negative normal form [Huang & Darwiche2005].

In this work, we present a novel approach that exploits database technology and in particular the structured query language (SQL). Previously, El Fattah [El Fattah1999] has introduced a framework for specification and representation of models in the language of relational databases. Adopting that representational framework we present in this paper a compositional approach to diagnosis based on a variable elimination algorithm.

Bringing model-based diagnosis to the realm of relational databases has several advantages. The modeling data typically resides in a database as in factory automation [El Fattah, Provan, & Darwiche1999] and representing models in the same language as the data enhances the integration of model-based diagnosis in real-life applications. Relational database management systems are robust and are designed to support very large amounts of data and thus can handle very large models. Data mining approaches are becoming more actively intertwined with relational databases and we expect to see this trend accelerating with the proliferation of data on the world wide web. Embedding model based diagnosis in the framework of relational databases enables integration with data mining tools for automated

construction of models from data and for automated induction of diagnostic rules from stored solutions of diagnostic problems.

Previously, El Fattah and Dechter [El Fattah & Dechter1995] have presented an approach to model-based diagnosis which formulates diagnosis as an optimization task in constraint networks. The approach uses tree clustering which involves transforming the original problem into a tree-like problem that can then be solved by a specialized efficient tree-solving algorithm [Dechter & Dechter1988, Dechter, Dechter, & Pearl1990]. The transforming algorithm identifies subproblems that together form a tree, and the solutions to the subproblems serve as the new values of variables in a tree metalevel problem. The metalevel problem is called a *join-tree*.

Variable elimination provides an alternative approach, with the same worst case complexity as tree clustering, which offers the advantage of interleaving the join tree transformation with the inference. In variable elimination the join tree is constructed incrementally while partial solutions are also computed. Using elimination we can partially evaluate the portions of the join tree that have small cliques and that can be solved without any space problem. This has practical implication for scaling-up diagnosis since real-life domains, e.g., digital circuits, are shown to have join trees the majority of whose clique sizes are relatively small while few cliques are distinctly large [El Fattah & Dechter1996].

The contribution of this paper is a compositional framework for model-based diagnosis embedded in the SQL language of relational databases. By explicitly representing the structure of the queries and their dependencies we are providing a meta language to adapt diagnostic inference from one problem to another. For instance, we can incrementally adapt our structured querying procedure to changes in the system description or to changes in the observation.

By separating the querying generation from their execution we can reuse parts of the diagnostic inference from one observation instance to another. Also, by formulating the elimination algorithm in terms of SQL queries we are leveraging all the query processing capabilities and query optimization techniques readily available in relational database management systems. For example, we can improve efficiency by exploiting known indexing techniques for query optimization and we can take advantage of the methods for computing the join of relational tables which are part of the SQL query engine.

The paper is organized as follows. First we describe our database representation of model-based diagnosis. Next we present a set of generic queries in SQL and describe our algorithms for query compilation and for processing observation. Next we provide discussion and related work followed by concluding remarks.

2 Database Representation

In this section we describe the representation of a model based diagnosis problem as an optimization query in a relational database.

Definition 1 [Relational Databases [Maier1983]] Let $U = \{U_1, ..., U_n\}$ be a set of attributes (variables), each with an associated domain. A *relational database scheme* $\mathbf{R}$ over U is a collection of relation schemes $\{R_1, R_2, \ldots, R_p\}$, where $\bigcup_{i=1}^{p} R_i = U$, and $R_i \neq R_j$, if $i \neq j$. A *relational database* Δ *on database scheme* $\mathbf{R}$ is a collection of relations $\{r_1, r_2, \ldots, r_p\}$. Each *relation* r_i on relation scheme R_i, written $r_i(R_i)$, is a set of value tuples $\{t_1, t_2, \ldots, t_p\}$ for the attributes in R_i from their respective domains. We write $t.X$ (or simply t_X) to denote the value assigned by a tuple t to the subset of variables $X \subset U$. A database Δ on scheme $\mathbf{R}$ over U entails a relation r on U defined as the join of all relations in Δ. That is r is the set of tuples $\{t = (U_1 = u_1, \ldots, U_n = u_n) | \forall j,\ t_{R_j} \in r_j\}$ Each tuple $t \in r$ is called a consistent solution for Δ.

Definition 2 [Weighted Relation] A weighted relation is a pair: (r, w), where r is a relation and w is a function defined on the scheme of r. That is, for each tuple $t \in r$ there is a weight assigned whose value is $w(t)$. A weighted relation (r, w) can be represented by a single relation whose scheme is $R \cup \{W\}$ where R is the scheme of r and W is a weight variable whose value is determined by the weight function w.

Following [de Kleer, Mackworth, & Reiter1992] we define *model-based diagnosis* as a triple: (SD, $COMPS$, OBS). The system description, SD, is a set of first-order sentences. The system components, $COMPS$, is a finite set of constants. A set of observations, OBS, is a set of first-order sentences. To formulate the diagnosis task as SQL querying, we map the triple (SD, $COMPS$, OBS) into a relational framework. The system description SD will be described in terms of two sets of variables: the *system variables* $X_1, ..., X_n$, which are the inputs and outputs of all components, with their associated finite domain values $dom(X_1), \ldots, dom(X_n)$, and the *assumption variables* $A = \{A_1, \ldots, A_m\}$. Each assumption variable A_j is associated with component $c_j \in COMPS$ and describes the component's functioning status. In the simplest case, these are bi-valued variables indicating whether the component is normal (value ok) or abnormal (value $faulty$). In the more involved case, they can index different fault models. Each component $c_j \in COMPS$ is associated with a relational table r_j describing its input-output behaviors under all its normal and abnormal conditions. Thus, the table r_j is defined over the scheme $R_j = \{A_j\} \cup S_j$, where S_j is the set of input and output

variables for component c_j. The observations OBS translate to forcing value assignments for a subset of system variables.

We augment the relation r_j for each component c_j by a weight (or cost) whose value is a non-negative function of the assumption variable A_j for the component. The function is zero if $A_j = ok$ and is positive if $A_j \neq ok$.

Given a model description and a set of observations, the diagnosis task is to construct an explanation, namely, an assumption tuple $(A_1 = a_1, \ldots, A_m = a_m)$ that can be extended to a solution $(X_1 = x_1, \ldots, X_n = x_n, A_1 = a_1, \ldots, A_m = a_m)$ consistent with the observations. The cost of an explanation is the sum of the costs associated with the assumption variables. That is,

$$w(\{A_1 = a_1, \ldots, A_m = a_m\}) = \sum_{A_j \in A} w_j(A_j = a_j). \quad (1)$$

If the cost has the same value for all non-ok modes then the minimal-cost diagnosis is the minimal-cardinality diagnosis.

Definition 3 [Model Description] A model description for model-based diagnosis is represented by a database Δ of weighted relations $\{r_j \mid j \in COMPS\}$ where $COMPS$ is the finite set of indexes for the system components. Each component j has a relation r_j defined over the scheme $R_j = \{A_j\} \cup X_j \cup \{W_j\}$. X_j is the set of input and output variables for the component. A_j is an assumption variable indicating the component's functioning status. W_j is a weight variable (or cost). In the simplest case, an assumption variable is a bi-valued variable indicating whether the component is normal (value ok) or abnormal (value $faulty$). In the more involved case, they can index different fault modes such as stuck-at-zero and stuck-at-one. A weight variable W_j is a non-negative function w_j of the assumption variable A_j for the component. The function is zero if $A_j = ok$ and is positive if $A_j \neq ok$.

Definition 4 [Diagnosis Problem] A diagnosis problem consists of a model description and a set of observations. The model description is defined on a set of system variables X, and assumption variables A. The observation is a value assignment for a set of observed variables $O \subset X$. We consider the model description as a database of weighted relations, $r_j(X_j, \{A_j\}, \{W_j\})$ for $j = 1, \ldots, m$, where the weight W_j for relation r_j is a non-negative function w_j of the assumption $A_j \in A$. $X_j \subset X$ is typically the input and output variables for a system component c_j. The observation is given by a relation $Obs(Var, Val)$ of variable-value pairs for each variable in O.

The diagnosis task is to construct an explanation, namely, an assumption tuple $(A_1 = a_1, \ldots, A_n = a_n)$ that can be extended to a solution

$(X_1 = x_1, \ldots, X_n = x_n, A_1 = a_1, \ldots, A_n = a_n)$ consistent with the observations. The cost of an explanation is the sum of the costs associated with the assumption variables. That is,

$$w(\{A_1 = a_1, \ldots, A_m = a_m\}) = \sum_{A_j \in A} w_j(A_j = a_j). \tag{2}$$

If the cost has the same value for all non-*ok* modes then the minimal-cost diagnosis is the minimal-cardinality diagnosis.

The diagnosis is a relation on the assumption variables defined as the projection on the assumption variables of the join of all model relations having minimum total cost. This can be stated by the following relational query,

$$\pi_A \min_{W_1 + \ldots + W_n} \bowtie_{j=1}^{n} \overline{r_j}(X_j, \{A_j\}, \{W_j\}) \tag{3}$$

Each $\overline{r_j}$ is obtained from the relation r_j and *Obs* by removing all tuples in r_j that assign values to variables inconsistent with *Obs*.

$$\overline{r_j} = \{t \in r_j \mid Y = X_j \cap O \wedge \forall j \ (Y_j, t_{Y_j}) \in Obs\} \tag{4}$$

Graph representation for a relational database can be constructed in two ways, as a *primal graph* or a *dual graph*. A primal graph represents variables by nodes and associates an edge with any two nodes residing in the same relation. A dual graph represents each relation by a node and associates a labeled arc with any two nodes that share variables. If the dual graph is a tree (called a join tree) or can be transformed into a tree by removing redundant arcs (in linear time), then the database is said to be acyclic [Maier1983]. In that case, a consistent solution can be assembled in linear time.

3 Example

This section is aimed to provide an informal description of our method using a realistic example of a circuit with fanout nodes, whose database model is not acyclic. Figure 1 shows the combinatorial circuit, c17, from the benchmark circuits [Brglez & Fujiwara1985] having 6 components, 5 inputs, and 2 outputs. The observed variables are the circuit's inputs and outputs. Each component is named by its output variable, and is associated with an assumption variable named by the letter "A" followed by name of the component. For example, "A10gat" is the assumption variable for the component whose output is "10gat". The weight parameter for each relation is labeled "Cost".

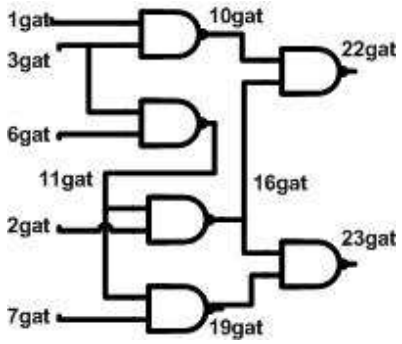

Figure 1: Circuit c17.

In the compilation phase we compute 4 types of queries: conditioning, combination, elimination and solution queries. We begin with the conditioning queries; each query applies to a relation whose schema variables include observed variables. We have 6 conditioning queries, one for each gate. The query, $cond\_10gat$, for gate $10gat$ is given by the SQL:

```
SELECT [10gat].[10gat], [10gat].A10gat,
        [10gat].Cost
FROM 10gat, Obs, Obs AS Obs_1
WHERE ((Obs.Var="1gat") AND
       ([10gat].[1gat]=[Obs].[Val]) AND
       (Obs_1.Var="3gat") AND
       ([10gat].[3gat]=[Obs_1].[Val]));
```

The above statement means to selects from the relation $10gat$ the tuples that are consistent with the observed values and projects the selection on the non-observed variables: $10gat$, $A10gat$ and $Cost$.

Each conditioning query modifies the database by replacing the conditioned relation by the relation computed by the query. After the conditioning phase is completed the elimination phase begins. The elimination phase processes the variables one by one in some designated ordering.

The elimination phase also modifies the database at each elimination step. An elimination step consists of collecting all the current relations that mention the eliminated variable, replacing them with their join relation and summing their weights, which is expressed by a combination query. If there is only one relaion that mentions the eliminated variable then the combination query is not needed. Next, an elimination query is comptued by projecting out the eliminated variable and keeping only the tuples with the minimum weight. A solution query is also computed which determines the optimal value of the eliminated variable for each value tuple of the remaining variables.

We consider the ordering; 11gat, 16gat, 19gat, A16gat, A19gat, 10gat, A22gat, A23gat, A10gat, A11gat, where the variable are eliminated in reverse order.

We first eliminate the variable A11gat. There is only one relation that mentions that variable, namely $cond_$11gat. The elimination query is:

```
SELECT cond_11gat.[11gat],
         Min(cond_11gat.Cost) AS Cost
FROM cond_11gat
GROUP BY cond_11gat.[11gat];
```

The solution query is:

```
SELECT cond_11gat.[11gat], cond_11gat.A11gat
FROM elim_A11gat, cond_11gat
WHERE((cond_11gat.[11gat]=[elim_A11gat].[11gat])
AND (cond_11gat.Cost=[elim_A11gat].[Cost]));
```

After the elimination phase is computed we end up with a set of solution queries that can be executed in the prescribed variable ordering.

The solution query for the first variable, 11gat computes a unary relation on that variable given by the SQL,

```
SELECT comb_11gat.[11gat]
FROM elim_11gat, comb_11gat
WHERE (comb_11gat.Cost=[elim_11gat].[Cost]);
```

The query simply selects the tuples in the combination query that has the minimum cost.

The next solution quey computes the value assignment for the second variable 16gat as function of the value of the first variable. All the solution queries will be such that when executed in the prescribed ordering, are guaranteed to construct a complete solution by repeatedly extending the partial solution one step at a time beginning with the first variable without backtracking.

The solution queries can be depicted by a dag, called solution dag. Each node of the dag represents a distinct variable and a value assignment can be computed by a solution query as a function of the value assignments of the parent variables for the node. The solution dag for the circuit c17 is shown in Figure 2

4 Compilation Approach

This section defines generic queries that perform basic inference operations. The generic queries will be used by our elimination algorithm to compile

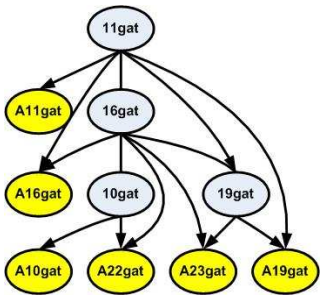

Figure 2: Solution dag for circuit c17.

system description into structured procedure for diagnosis. We assume familiarity with the structure of the structured query language (SQL).

4.1 Conditioning query

The query $cond(r; X)$, written $\delta_{X=x}\ r$, takes a weighted relation $r(R \cup W)$ and a subset $X \subset R$ and returns the relation obtained by selecting only tuples from r whose X value equals a parameter x and by projecting onto $R \setminus X$,

$$SELECT\ r.[R \setminus X]\ FROM\ r\ WHERE\ X = x$$

4.2 Combination query

The query $comb(\{r_1, \ldots, r_k\})$, written $r_1 \otimes \ \ldots \otimes r_k$, takes $k \geqslant 2$ weighted relations $r_1(R_1 \cup W), \ldots, r_k(R_k \cup W)$ and combines them into a weighted relation obtained by joining the k relations and summing their weights. Without loss of generality the combination query for $k = 2$ is as follows:

$$\begin{aligned} &SELECT\ r_1.R_1, r_2.[R_2 \setminus R_1], r1.W + r2.W\ As\ W \\ &\quad FROM\ r_1,\ r_2 \\ &\quad WHERE\ r1.(R_1 \cap\ R_2) = r2.(R_1 \cap\ R_2) \end{aligned}$$

4.3 Elimination query

The query $elim(r, X)$, written $\oplus_X\ r$, takes a weighted relation $r(R \cup W)$ and a subset $X \subset R$ and returns a new relation obtained by projecting out X and minimizing the cost.

$$\begin{aligned} &SELECT\ r.[R \setminus X], Min(W) \\ &\quad FROM\ r\ GROUP\ BY\ r.[R \setminus X] \end{aligned}$$

Algorithm: compile-SP
Input: relational schema of the system description and the observables
Output: SP represented by a dag G.
initialization: L is the set of associations between variables and relations; G is empty.

Conditioning: For each relation $r_i(R_i)$ whose scheme includes observed variables O_i create a conditioning query $cond(r_i; O_i)$ and replace all associations for r_i in L by those of the conditioning relation.

Elimination: Until L is empty do: (i) select a variable X_i having minimum number of neighbors; i.e., variables associated with same relations as X_i in L; record X_i in an ordering table. (ii) if X_i is associated with a set $\{r_{i1}, \ldots, r_{ik}\}$ of two or more relations then create a combination query $s_i = \bigotimes_{j=1}^{k} r_{ij}$; add s_i to the nodes of the dag and add edges directed from each r_{ij} to s_i else s_i is the one relation associated with X_i. (iii) create an elimination query $r_e = \bigoplus_{X_i} s_i$ and add it to the nodes of the dag and add an edge from s_i to r_e. (iv) create a solution query $x_i^* = sol(s_i, X_i)$ add it to the dag's node and add two edges from r_e and s_i to x_i^*. The variables other than X_i in the relation x_i^* are the parents $pa(X_i)$. (v) update the association L by deleting all entries on X_i and adding new associations between $pa(X_i)$ and the elimination relation r_e.

Diagnosis: First, compute an answer query q_A by proceeding in reverse elimination ordering: (i) $q_A \leftarrow \emptyset$; (ii) For $i = n$ downto 1 do $q_A \leftarrow q_A \bowtie x_i^*$; Then, compute a diagnostic query that selects the assumption variables from q_A.

Figure 3: Algorithm compile-SP.

4.4 Solution query

The query $sol(r, X)$ takes a weighted relation $r(R \cup W)$ and returns the functional relation $x^* \mid_r : (R \setminus X) \rightarrow X$ obtained by computing the value x^* of X in relation r having minimum weight given the value of $R \setminus X$. A solution query requires the elimination subquery, $r_e = elim(r, X)$ and is given by the SQL expression:

$$SELECT\ r.R\ FROM\ r_e,\ r$$
$$WHERE\ r_e.W = r.W\ AND\ r_e.[R \setminus X] = r.[R \setminus X]$$

5 Query Compilation

In this section we present a compilation algorithm for computing structured procedures (SP) in SQL for diagnosis. The algorithm is based on variable elimination and takes as input a schema of the system description and the observation and outputs a dag whose nodes are instantiations of generic queries and whose edges determine execution precedence for the queries.

The compile-SP algorithm in Figure 3 consists of 3 main steps: conditioning, elimination, and diagnosis. The conditioning step eliminates all the observable variables by applying a conditioning query to each relation whose scheme includes observable variables. The elimination step eliminates the non-observable variables one by one. The elimination step also creates solution tables for the eliminated variables. The solution tables are represented by a solution dag whose set of families (nodes and their direct parents) are the schema for the tables. The diagnosis step computes an answer query that joins the solution tables then projects the answer on the set of assumption variables.

The algorithm maintains the information on the changing database schema during elimination in a table that records current associations between variables and relations. The output of the algorithm is a structured querying procedure represented by a dag that can be executed for any given observation instance.

Example 1 The compiled procedure dag for circuit c17 computed by the compile-SP algorithm is shown in Figure 4. The root nodes of the dag are the conditioning queries and the internal nodes are elimination, combination, and solution queries. Elimination of the assumption variables required no combination queries since each assumption resides in only one component. The non-observables shared in the schema of more than one relation require combination queries prior to elimination. We note that eliminating $16gat$ required combining 3 relations instead of 2, indicating added edges in the process of elimination due to the fact that the database is initially cyclic. The ordering of elimination is $11gat$, $16gat$, $19gat$, $A16gat$, $A19gat$, $10gat$, $A22gat$, $A23gat$, $A10gat$, $A11gat$. The solution dag for the circuit is shown in Figure 2

6 Processing Observations

In this section we describe an algorithm for processing observation given the compiled SP. The algorithm is shown in Figure 5. The algorithm uses

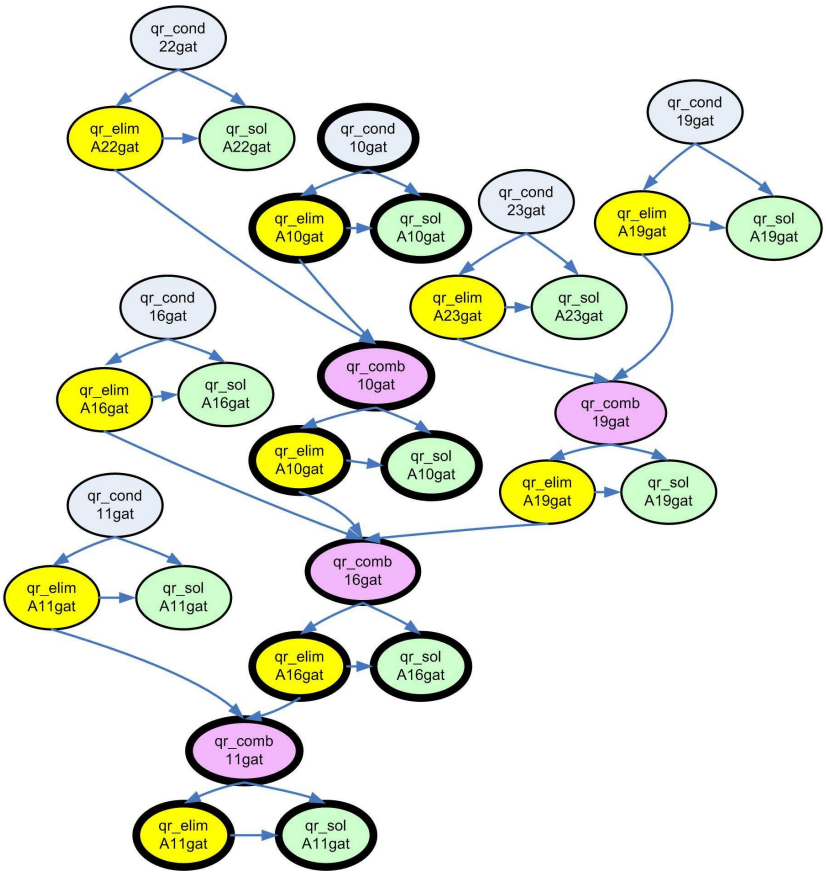

Figure 4: SP graph for circuit c17.

the structure of the SP represented as a dag. The algorithm caches the tables computed by the queries and updates the tables only if needed when processing next observations. The key idea is that the diagnosis answer is *sensitive* only to changes in the input conditioning tables which may not change for multiple observation tuples. For example, the nand gate 10gat in Example 3 will not change the conditioning table for 3 observation tuples where one of the inputs is *false* Thus the conditioning input tables to the SP can be viewed as "triggers" for incrementally updating the diagnosis answer to adapt to changes in the observation. If there is no modification in the input tables then there is no change in the diagnosis answer. If the observation produces triggers then the process-OBS will execute only part of the SP determined by a local propagation of the triggers through the dag. Checking whether a new observation changes the current conditioning table is equivelent to checking equality of two relations.

Algorithm: process-OBS
Input: SP as a dag G; observation tuple OBS
Output: One (All) explanations or diagnosis
/*First we check if the observation creates new triggers then execute only part of the procedure to update tables*/

1. for each root (conditioning) node in SP
2. if the new conditioning table is different from the previous table
3. then mark the query as active
4. Until G is empty do:
   (a) Determine all the root nodes N_{root} in G
   (b) for each node in N_{root} do:
   (c) if the node is active then
   (d) execute the query for the node
   (e) mark all the children as active
   (f) remove the node from G.
5. Execute the solution or diagnosis query.

Figure 5: Algorithm process-OBS

Example 2 Executing the SP for circuit c17 on the observation where all inputs are *false* and all outputs are *true* will yield one single fault diagnosis, namely the gate $16gat$ is $sa0$. Applying process-OBS on a new set of observation where only the input $1gat$ is changed to *true* we get no activation and the diagnosis will remain the same as before. Applying process-OBS on a new set of observation where the input $3gat$ is also modified to *true* with the rest remaining the same we get one trigger: qr_cond_10gat. The propagation of the trigger and the part of the SP that will get executed by process-OBS is highlighted on the dag in Figure 4. Only 12 out of 30 tables will get updated by executing the 12 highlighted queries of the SP. The diagnosis for the new observation consists of 3 single faults: $A23gat = sa1$ or $A19gat = sa0$ or $A16gat = sa0$.

7 Discussion and Related Work

Our approach is based on the general technique of variable elimination in non-serial dynamic programming [Bertelè & Brioschi1972] formulated in a relational framework similar to bucket elimination [Dechter1996]. Our approach differs from bucket elimination in that we express the elimination steps explicitly in relational algebra which enables execution on any database query engine and in that we separate the compilation of elimination procedure from their execution phase so that compilation can be done prior to actual observation. The worst-case time and space complexity of our compiled procedure is $O(exp(w^*(d))$ where d is the min-degree ordering [Bertelè & Brioschi1972] (page 55) and w^* is the induced width [Dechter & Pearl1989] for the ordered primal graph after removing the observable nodes and their incident edges. This worst case complexity may not reflect the actual average case performance of our algorithm and is expected to improve by our caching and reuse of inference across observations. The compilation approach presented in [Darwiche1998] is similar to ours in that it exploits the structure of the device and has the same worst case complexity; namely exponential in the tree width. Our approach differs from [Darwiche1998] in both what is compiled off-line and what is processed on-line. What we compile off line is a SQL-querying procedure that depends only on the structure of the system description while [Darwiche1998] computes a "compiled system description", namely a logical sentence equivalent to eliminating the non-observables. At observation time [Darwiche1998] computes the diagnosis from the compiled system description by performing conditioning and optimization which in our approach are already included in the compilation phase. Standard approach to model-based diagnosis [Reiter1987] is based on the notion of conflicts which are used to guide the search in the diagnostic space. Conflicts are assumption tuples that have no solutions, i.e., they are inconsistent with the system description and the observation. A variation of our querying procedure can be computed to form a correct and complete consistency checker that can be used in combination with the conflict based approach to diagnosis [Mauss & Sachenbacher1999]

8 Concluding Remarks

In this paper we present a structure-based approach to diagnosis embedded in the language of relational databases. The approach compiles the schema of the system description and the observation to a structured procedure represented by a dag whose nodes are SQL queries and edges denote

execution precedence. Our approach is suitable for compiling efficient procedures from device descriptions to perform onboard model-based diagnostics. Our approach speeds up the diagnostics in two ways. One, the compilation procedure executes off-line to compute a structured querying procedure that exploits the problem structure and SQL query optimization techniques. Two, the procedure execution at observation time exploits inference made on previous observations to avoid expensive recomputation. If the time available for diagnostic reasoning onboard is stringent we can run our compiled procedure off-line on all instantiations of the observable variables and record the results in a database. This is feasible using our approach since diagnosis is computed once for each observation pattern defined as sets of observation tuples having same conditioning tables.

References

[Barrett2005] Barrett, A. 2005. Model compilation for real-time planning and diagnosis with feedback. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence(IJCAI-05).*

[Bertelè & Brioschi1972] Bertelè, U., and Brioschi, F. 1972. *Nonserial Dynamic Programming.* New York: Academic Press.

[Brglez & Fujiwara1985] Brglez, F., and Fujiwara, H. 1985. A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran. Proc. IEEE Int. Symposium on Circuits and Systems. distributed on a tape to participants of the Special Session on ATPG and Fault Simulation, Int. Symposium on Circuits and Systems, June 1985; partially characterized in F. Brglez, P. Pownall, R. Hum, Accelerated ATPG and Fault Grading via Testability Analysis.

[Darwiche1998] Darwiche, A. 1998. Compiling devices: A structure-based approach. In *International Conference on Principles of Knowledge Representation and Reasoning (KR-98),*, 156–166.

[de Kleer, Mackworth, & Reiter1992] de Kleer, J.; Mackworth, A.; and Reiter, R. 1992. Characterizing diagnoses and systems. *Artificial Intelligence* 56:197–222.

[Dechter & Dechter1988] Dechter, R., and Dechter, A. 1988. Belief maintenance in dynamic constraint networks. In *Proceedings, AAAI-88*, 37–42. Menlo Park, CA: AAAI Press/ The MIT Press.

[Dechter & Pearl1989] Dechter, R., and Pearl, J. 1989. Tree clustering for constraint networks. *Artificial Intelligence* 38:353–366.

[Dechter, Dechter, & Pearl1990] Dechter, R.; Dechter, A.; and Pearl, J. 1990. Optimization in constraint networks. In Olivier, R., and Smith, J., eds., *Influence Diagrams, Belief Nets and Decision Analysis*. New York: J. Wiley. 411–425.

[Dechter1996] Dechter, R. 1996. Bucket elimination: A unifying framework for probabilistic inference. In Horvitz, E., and Jensen, F., eds., *Uncertainty in Artificial Intelligence*. Morgan Kaufmann. 211–219.

[El Fattah & Dechter1995] El Fattah, Y., and Dechter, R. 1995. Diagnosing tree-decomposable circuits. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*.

[El Fattah & Dechter1996] El Fattah, Y., and Dechter, R. 1996. An evaluation of structural parameters for probabilistic reasoning: Results on benchmark circuits. In *Uncertainty in Artificial Intelligence (UAI-96)*, 244–251.

[El Fattah, Provan, & Darwiche1999] El Fattah, Y.; Provan, G.; and Darwiche, A. 1999. Model based diagnosis for factory automation: Challenges and open problems. In *Working Notes of the Tenth International Workshop on Principles of Diagnosis (DX99)*. 68–77.

[El Fattah1999] El Fattah, Y. 1999. Structured modeling language for automated modeling in causal networks. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, 1108–1114.

[Huang & Darwiche2005] Huang, J., and Darwiche, A. 2005. On compiling system models for faster and more scalable diagnosis. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, 300–306.

[J.de Kleer1990] J.de Kleer. 1990. Compiling devices and processes. In *Fourth International Workshop on Qualitative Physics*.

[Maier1983] Maier, D. 1983. *The Theory of Relational Databases*. Rockville, MD: Computer Science Press.

[Mauss & Sachenbacher1999] Mauss, J., and Sachenbacher, M. 1999. Conflict-driven diagnosis using relational aggregations. In *Working Notes of the Tenth International Workshop on Principles of Diagnosis (DX99)*. 174–183.

[Reiter1987] Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32:57–95.